Antonio Roberto Bellarmino

THE HYBRID BUSINESS ANALYST

A Pragmatic Guide to Requirements, Strategy, and Real-World Project Success.

Contents

1 Introduction **9**
- 1.1 Who should read this book? . . . 10
- 1.2 How to facilitate meetings . . . 11
- 1.3 What a requirement is . . . 12

2 The Step 0 **13**

3 The WHY question, Step 1 **15**
- 3.1 Stakeholders identification . . . 17
- 3.2 Project Initial Scope . . . 19
- 3.3 Gap Analysis . . . 19
- 3.4 Benefits . . . 20
- 3.5 Project Overview . . . 21
- 3.6 Success Factors . . . 21
  - 3.6.1 Financial Return Indicators . . . 22
- 3.7 Costs . . . 22
- 3.8 Risks . . . 23
  - 3.8.1 Risk Identification . . . 23
  - 3.8.2 Assumptions . . . 24
  - 3.8.3 Regulatory and Compliance Considerations . . . 24
  - 3.8.4 Risk Mitigation Strategy . . . 24
- 3.9 Business Requirements . . . 25
  - 3.9.1 Distinguishing Needs from Requirements . . . 25
  - 3.9.2 Business Epic Analysis . . . 26
  - 3.9.3 MoSCoW Prioritization . . . 27
- 3.10 Features Delivered for Each Release . . . 27

4 The What and How questions, step 2 **29**
- 4.1 Use Cases: The Core Tool . . . 30
  - 4.1.1 Briefly described Use Case . . . 30
  - 4.1.2 Fully described Use Case . . . 31
  - 4.1.3 Use Case Scenario Steps . . . 32

4.1.4 Visual Support 33
4.2 Main Flow, Alternate Flows, and Exceptions 34
4.2.1 The Three Flow Types 34
4.3 Business Rules in Parametric Form 35
4.4 Use Case Example 36
4.5 Non-functional Requirements 37
4.5.1 Examples of Testable NFRs 37
4.5.2 Security Requirements 38
4.5.3 Global Non-Functional Requirements 39
4.5.4 Prioritizing Non-Functional Requirements 39
4.5.5 Example: NFRs for Course Booking 39
4.5.6 Key Insight for the BA 40

5 Change Requests **41**
5.1 The Change Control Process 41
5.2 Best Practices for Change Management 41

About the Author **43**

List of Tables

1.2 Mapping of Strategic Questions to Business Analysis Deliverables . . 10

3.1 Example of a Stakeholder Analysis Table for a Project 17
3.2 Example of a User Roles Table 18
3.3 Gap Analysis Example 20
3.4 Project Cost Breakdown (Example) 23
3.5 Requirement Traceability Table 26
3.6 Release Plan and High-Level Implementation Schedule 28

4.3 Prioritized NFRs for the Course Booking use case. 40

List of Figures

3.1 Power-Interest Grid for Stakeholder 18

4.1 Wireframe example for the 'Book Course' use case. 34

Copyright Page

The Hybrid Business Analyst:

A Pragmatic Guide to Requirements, Strategy, and Real-World Project Success

Copyright © 2025 Antonio Roberto Bellarmino
All Rights Reserved.

Electronic Edition License:
This ebook is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0).

You are free to share, copy, and redistribute the material in any medium or format under the following terms:

- **Attribution:** You must give appropriate credit, provide a link to the license, and indicate if changes were made.
- **Non-Commercial:** You may not use the material for commercial purposes.
- **No Derivatives:** If you remix, transform, or build upon the material, you may not distribute the modified material.

To view a copy of this license, visit:
`http://creativecommons.org/licenses/by-nc-nd/4.0/`

Chapter 1

Introduction

This book is inspired by [1] While I value the structure and clarity of that work, my aim here is to go beyond it — adding my personal insights and lessons learned from real projects.

In my career, I have seen projects succeed and fail under both Waterfall and Agile. I don't believe in choosing one over the other as an article of faith. Instead, I believe in borrowing the best elements of each — a hybrid mindset that respects structure when it's needed, but remains flexible when change is inevitable.

For instance, why not use a User Story as the main descriptor of a Use Case? Both are powerful tools—so why not combine them to create something even more effective?

However, do not take this wrong: I am not saying "do whatever you want and mix everything". No! What I am saying is to take advantage of the whole landscape of tools that a business analyst has.

My approach is targeted: when a requirement involves human intervention—which is the case for most user-facing features—I prefer using User Stories to define the use cases. However, when the actors are exclusively systems, I pivot to more technical specifications or sequence diagrams better suited for back-end integrations.

Before we go further, let's be clear about what this book is not.

I won't teach you UML, BPMN, flow charts, or Gherkin syntax from scratch — you should already be familiar with these. I will, however, include examples of them when they are useful for illustrating a point.

The core of Business Analysis is about answering the essential questions that drive business analysis, and understanding the documents and techniques that bring those answers to life:

- WHY does the organisation need this new project or change to an existing one? Yayici (2015)
- WHO will use the system? This can be optional if the list of users is small. In such cases, clearly state in the use cases who the intended user is. (This

is my addition to Yayici's framework. In most cases, it is crucial to explicitly identify the users. If the list of users is small, this step can be optional, provided the intended user is clearly stated in the use cases.)

- According to Yayici (2015) WHAT must the users and/or the system do? Yayici (2015)
- According to Yayici (2015) HOW will the users and/or the system perform what was described in the previous point? Yayici (2015)
- According to Yayici (2015) there is another question to be answered, "TECHNICALLY HOW", but this won't be tackled in this book.

The answers to these questions are captured within specific deliverables produced by the Business Analyst and they are the ones listed in this table:

Questions	**Document**
Why?	**Initiation document** (also known as Business Requirement Document). In Agile jargon, this is related to the "Theme".
Who?	High-Level Use Case diagram(s).
What and How?	The document name varies by organization (private or public), but the most common are: • System Requirements Specifications • Use Case Specifications • Functional Requirement Document In this text, we focus on the **Use Case Specifications document**. Depending on size, this relates to a Business Epic or a User Story in Agile.
Technically how?	**Architects' Responsibility:** • ER Diagram, Class Diagram, etc.

Table 1.2: Mapping of Strategic Questions to Business Analysis Deliverables

1.1 Who should read this book?

If you are new to business analysis, this book will help you avoid the most common early mistakes — from writing documents no one reads to gathering requirements nobody needs.

If you are an experienced BA, you'll find techniques, templates, and war stories that can save you weeks of rework and help you handle stakeholders with confidence.

If you are somewhere in between, you'll gain a practical framework for when to be structured, when to be flexible, and how to keep a project moving forward without getting lost in methodology debates.

This is not a theory book. It is a field guide for Business Analysts who want to deliver value, reduce friction, and make both users and sponsors feel that the project is in safe hands.

1.2 How to facilitate meetings

Before we can document a single requirement, we must master the art of the conversation.

As a Senior Business Analyst, you serve as the bridge between stakeholder vision and technical execution.

Here, you have a must-have framework to facilitate high-stakes meetings. It is not merely an administrative task; it is a strategic intervention designed to extract clarity from complexity.

- **Stakeholder Curation:** You proactively identify and engage the necessary participants, ensuring that each person arrives with a clear understanding of their role—whether they are there to provide expertise, represent a user group, or provide formal sign-off.

- **Defining the Strategic Purpose:** You establish the "Why" immediately to set the stage for productive dialogue.
  Example: You frame the session to uncover specific workflow disruptions caused by the introduction of a new product type.

- **Driving Toward Tangible Objectives:** You define what success looks like before the meeting starts, ensuring the team moves toward a concrete finish line.
  Example: By the end of the call, you have finalized the impact map for Customer Service and secured a commitment for the next discovery phase.

- **Value-Oriented Agenda Design:** You frame topics through the lens of stakeholder benefits, using targeted questions to drive the conversation toward critical decisions and scope boundaries.

- **Decision Governance:** You actively identify when a decision is required, ensuring that outcomes are documented and that the project avoids the trap of circular discussions.

1.3 What a requirement is

First, we must clarify the definition of a requirement. It is a term often mistaken for a need, yet distinguishing between the two is critical for project success.
When you, as a Business Analyst, interview stakeholders, they will usually express themselves in terms of needs, for example:

"The system must be faster."
"This page must be configurable."

These statements are not requirements. They are needs — general expressions of a desired outcome.
A requirement goes further: it is a precise, testable, and traceable statement that fully describes what must be delivered to meet that need.

A well-written requirement should:

- Be unambiguous
- Depict the whole need concisely and to the point
- Be testable (measurable)
- Be prioritised
- Be categorised (e.g., business requirement, functional requirement, non-functional requirement)
- Be traceable
- Be atomic (cannot be broken down further without losing meaning)

In practice, a good starting point is to focus on three essential qualities:

- Unambiguous
- Atomic
- Testable

For example, the two needs mentioned earlier could be rewritten as requirements in this way:

- *The system shall display the search results within 2 seconds for 95% of queries containing up to 50 results.*
- *The "Settings" page shall allow administrators to configure the following parameters without modifying the source code: [list parameters].*

Chapter 2

The Step 0

Many Business Analysts (BA) are taught to begin with business needs. While logical, this approach is incomplete. The real first step is to understand the organization's mission, why, and vision (MWV). If you already know them—review them. You cannot deliver meaningful analysis without first grasping why the organization exists, how it operates, and where it intends to go. Do not make the mistake of overlooking this step. Often, these foundational elements are unclear even to the organization itself; as a Business Analyst, you must dig deep to find—or help define—them properly.

When I joined a public institution, I rushed into analysing a new feature for their key application. Soon, I realized I'd skipped the most important step: understanding the institution's purpose and structure. Without that context, I risked heading in the wrong direction. I went back to square one—asking questions, researching, and mapping the organization's landscape. Only then was I ready to begin analysis.

Exploring MWV can reveal even deeper insights: sometimes the mission and vision are outdated or misaligned with current operations. Here, a BA can add value immediately by helping the organization revisit its foundations.

The goal is always to bring innovation, improvement, and value; "people don't buy what a company does; they buy why it does it." (cit. Simon Sinek, 2013, `https://www.youtube.com/watch?v=qp0HIF3SfI4`)

Chapter 3

The WHY question, Step 1

The first step of any business analysis effort is to understand the **"Why"** behind the project—the rationale that justifies its existence. As previously mentioned, once you've clarified the broader context or "universe" in which the organization operates, your next move is to uncover the specific reasons that led to the request for a new project or functionality.

This rationale should be captured in a **Business Case Document**, also referred to as an Initiation Document. It represents the **outermost layer** of the documentation "onion" you'll build throughout your analysis work. These layers typically include (but are not limited to):

- **Business Case Document**. A well-crafted Business Case Document is the cornerstone of every successful project. From a Project Management perspective, it is the ultimate source of truth—the document that justifies every decision and anchors the project to its strategic purpose.
- **Use Case Document**, describing:
  - User goals
  - Functional requirements
  - Non-functional requirements
- **Technical Documentation:** This layer involves the underlying system architecture and data structures. While these are primarily the responsibility of Solution Architects and Developers, they are a critical reference point for the Senior BA.
  - Examples: Entity-Relationship (ER) Diagrams, Class Diagrams, and API Specifications.
  - **The BA's Role:** While the BA doesn't "own" or create these technical artifacts, he/she must understand them deeply to ensure that the busi-

> ness "What" aligns perfectly with the technical "How." A gap between these two layers is where most project failures begin.

Understanding the real problem or business need—which you'll translate into Business Requirements—ensures that you're delivering maximum value to your stakeholders. Your goal at this stage is to define why the business request is needed: in other words, to understand the rationale behind the business need.

To determine the appropriate level of documentation, I find it useful to categorize projects based on their estimated duration in **Sprints** (assuming a standard 2-week cycle). This approach, inspired by the principles of scaled analysis[1], helps balance governance with agility:

- **Large-scale Initiatives (12+ Sprints):** this rationale should be formally documented in a Business Case Document, detailing the business requirements.
- **Tactical Projects (3 to 12 Sprints):** For mid-sized efforts, a **Vision and Scope Document** is usually sufficient to define the boundaries and expected outcomes without excessive overhead.
- **Agile Enhancements (1 to 2 Sprints):** For minor modifications or "quick wins," a **one-page summary** (or a well-defined User Story with clear acceptance criteria) provides enough context for the team to act.

Typical business requirements may include:

- A process the organization must carry out
- A piece of data needed to support that process
- A business rule governing the process or data
- A known issue and/or its root cause

Regardless of project size, a preliminary analysis is essential to identify the primary problem and, where possible, its root cause. This involves mapping the current ("as-is") processes, identifying pain points, and proposing solutions to existing challenges—whether the initiative is a brand-new project or an update to an existing one.

Let's now examine how to structure this document. For every project, you should determine which of the following sections are necessary to include in your Initiation Document.

[1] This classification adapts the duration-based model found in [1] to an Agile context.

3.1 Stakeholders identification

It is important to remember that a stakeholder is not necessarily a user of the system. A stakeholder is anyone who has an interest in or influence over the project—this may include sponsors, business managers, project owners, and others. While literature abounds with detailed stakeholder classifications, in this context we will focus on what is relevant from a Business Analyst's (BA) perspective.

It is vital to distinguish between a Stakeholder and an Actor. A stakeholder has an interest in the project's success, whereas an actor (or User Role) interacts directly with the system. As a BA, you must map both: the people who provide the requirements and the roles that will execute them.

From the BA's point of view, the following information should be collected for each stakeholder:

- Name
- Role in the project
- Responsibilities (i.e., what decisions they are entitled to make)
- Preferred communication method (e.g., email, meetings, messaging apps)

You are, of course, free to expand this list depending on the project's needs or your organization's standards.

While this step may be optional for very small-scale projects, it is generally essential to identify all relevant stakeholders—especially the end users who will benefit from the system. Although this details may be documented either in a separate **Stakeholder Register** or directly within the **Use Case Specifications**, I prefer the former.

If you have already developed a **High-Level Use Case Diagram**, you can supplement it with a stakeholder table like the example below, to provide a complete picture of the project's human landscape:

Stakeholder Name	Role	Responsibilities	Contact Method
John Smith	Business Owner	Final decision on requirements	Email / Weekly call
Jane Doe	Project Sponsor	Budget approval	Email
Alice Brown	End User	Feedback on usability and needs	In-person workshop

Table 3.1: Example of a Stakeholder Analysis Table for a Project

A similar table can be created for the user roles:

User role/Actor	Mandatory	Action	Internal / External
admin	Yes	Full system configuration, user management	Internal
manager	yes	Approve budgets, assign tasks	Internal
Guest user	No	View limited public content	External
Customer	Yes	Place orders, track shipments	External

Table 3.2: Example of a User Roles Table

Identifying names and roles is merely the starting point. To manage your time and project expectations effectively, you must categorize stakeholders based on two key dimensions: **Power**—the ability to influence or block the project—and **Interest**—the extent to which they are affected by the outcome.

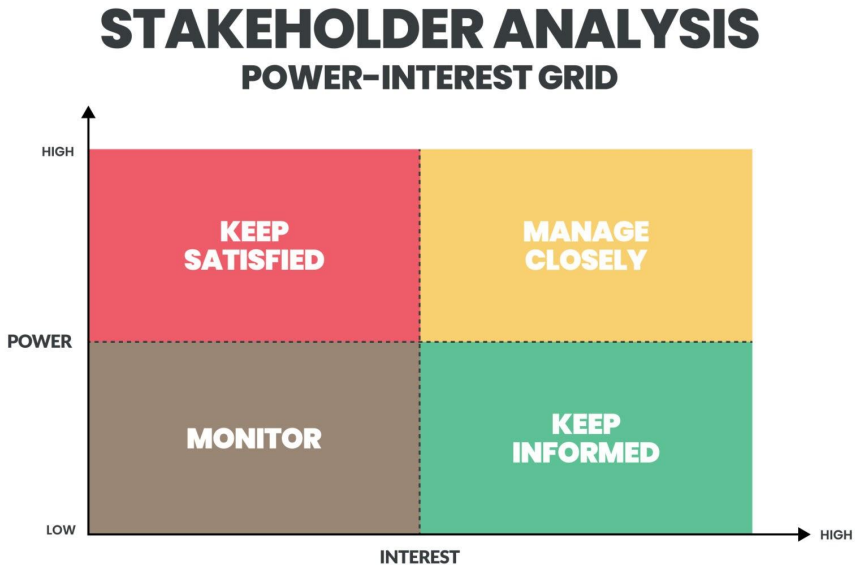

Figure 3.1: Power-Interest Grid for Stakeholder

3.2 Project Initial Scope

The **Project Initial Scope** is also named "Major Features." The initial scope of the project should, whenever possible, be described from the end users' point of view. A user-centric approach ensures that the solution remains aligned with real needs and expectations.

One effective way to express requirements from this perspective is through **User Stories**—short, structured narratives that describe what a user wants to achieve and why. At this early stage, these stories are often too broad to be implemented directly and are therefore classified as **Business Epics**. As a general guideline, if a story cannot be completed within a single sprint, it should be categorized as an Epic. These Business Epics will later be decomposed into smaller, actionable user stories.

There are several approaches to capturing features and requirements, but two formats are particularly useful:

- **VERB + NOUN pattern**: Examples include *book courses*, *manage teachers*, *generate reports*. This format is versatile and works well for both user-centric and system-centric analysis.

- **User Stories**: Structured as "As a [type of user], I want to [action] so that [benefit]." This format clarifies both the functionality and the intended value.

To further define the initial project scope and identify which stakeholders (or user groups) will interact with the system, I recommend creating a **High-Level Use Case Diagram** using UML (Unified Modeling Language). This diagram visually represents the key actors and their goals, offering a concise overview of system boundaries and main functionalities.

This early visualisation serves as a foundation for detailed use case development and ensures alignment among all parties involved in the project.

3.3 Gap Analysis

It is common practice to include a **Gap Analysis** in the Business Case document. A Gap Analysis compares the current state ("as-is") of a process or system with the future state ("to-be") expected after the project's implementation. This comparison highlights the changes required to bridge the gap and achieve the desired outcomes.

Actor	Goal	Current State	Future State
Teacher	Create new training	Manually managed with external tools	Training administrator console
User	See overview of trainings	Legacy App	Smart Portal

Table 3.3: Gap Analysis Example

Best practice: Create a high-level diagram outlining the scope of the solution. **UML Use Case Diagrams** are particularly effective here, as they focus on each actor and clearly define their main goals. Remember: a use case represents the goal of a specific actor—not a technical function of the system.

Additionally, **User Journeys** can be valuable at this stage. When combined with mockups, they help visualise both the pain points in the current process and the benefits introduced by the new solution.

User Journeys are a powerful technique for gathering and validating requirements. A User Journey (UJ) is a sequence of steps that a user takes to achieve a specific goal:

- **AS-IS Analysis**: UJs can be used to map the current process and identify "pain points," bottlenecks, or inefficiencies.
- **TO-BE Design**: UJs help describe the future state and how the proposed solution addresses the issues identified.

It is highly effective to represent a User Journey through **wireframes**. This allows you to visualize how the interface appears to the user at each step. By doing so, you can pinpoint exactly where a user might face difficulties or, conversely, identify where the functionality is particularly effective.

Once the gaps between the current and future states are clear, the next logical step is to articulate the value of closing those gaps. This is where the benefits of the project come into focus.

3.4 Benefits

To effectively describe the benefits that the new project will deliver, it is useful to address the following key questions:

What problems will the project solve?

Identify the specific pain points, inefficiencies, or risks that the project will address.

What opportunities will it enable?
Highlight the new possibilities, markets, or efficiencies that the solution will unlock.

Why is the proposed solution important?
Explain its relevance in terms of business strategy, compliance, competitiveness, or operational excellence.

Who will benefit, and in what way?
Specify the stakeholders or user groups impacted and describe how they will experience improvements.

Answering these questions will help clearly define the value proposition of the project, demonstrating both its role in resolving existing issues and its potential to generate strategic advantages.

While benefits describe the "why" in terms of value, the Project Overview sets the stage for the initiative at a higher level.

3.5 Project Overview

The **Project Overview** is a key section of the Business Case, as it sets the stage for the entire initiative. It provides a high-level understanding of:

- **Vision** – a forward-looking, idealistic description of the desired long-term outcome.
- **Objectives** – specific and measurable goals the project aims to achieve. This ensures that progress can be tracked and results can be evaluated objectively.
- **Context** – the environment, constraints, and circumstances in which the project will operate, including any dependencies, risks, or organisational factors that may influence its execution.

A clear and concise project overview ensures that all stakeholders share a common understanding of the project's intent and strategic alignment.

3.6 Success Factors

A **Critical Success Factor (CSF)** is a key element that directly supports the achievement of a business strategy. For example, if the strategic goal is to increase the customer base, a CSF could be to grow the number of customers by a specific percentage within a set timeframe. Success factors should be clearly measurable and aligned with the organization's overall objectives to ensure they provide meaningful guidance.

Financial return indicators, such as revenue growth, profit margins, or **Return on Investment (ROI)**, are commonly used success factors because they quantify the economic impact of a project or initiative. However, success factors can also include customer satisfaction scores, market share growth, or operational efficiency improvements, depending on the strategic priorities.

It is essential to continuously monitor these success factors throughout the project lifecycle. Regular tracking enables timely adjustments, helps identify potential risks early, and ensures that the project stays aligned with its intended outcomes. By focusing on well-defined success factors, business analysts and project teams can better demonstrate the value delivered and contribute to informed decision-making.

3.6.1 Financial Return Indicators

To evaluate whether a project is financially viable, it's important to measure its expected return. The most commonly used financial return indicators include:

Return on Investment (ROI): Measures the overall profitability of the project.

$$\text{ROI} = \left(\frac{\text{Net Benefit}}{\text{Cost}} \right) \times 100$$

Payback Period: The time required to recover the initial investment from the net benefits generated. Shorter payback periods are usually preferred.

Net Present Value (NPV): Calculates the present value of all future cash flows (inflows and outflows), discounted over time. A positive NPV indicates a profitable project.

Internal Rate of Return (IRR): The discount rate at which the NPV becomes zero. A project is considered attractive if the IRR is higher than the entity's required rate of return.

These indicators help stakeholders compare different initiatives, prioritize investments, and make informed decisions based on both strategic and financial value.

3.7 Costs

Understanding and accurately estimating the costs of a project is essential for assessing its feasibility and justifying the required investment. This section should cover both initial and ongoing costs, such as:

- **Development and implementation**: including internal resources, external consultants, and infrastructure setup.

- **Licensing or software acquisition**: purchase or subscription fees for any required tools or platforms.
- **Training and change management**: resources needed to prepare users and support organisational adoption.
- **Operational and maintenance**: recurring expenses to ensure the solution remains functional, secure, and up to date.

Where possible, provide quantitative estimates or ranges, specifying the calculation method and data sources used. For large-scale initiatives, include a detailed cost breakdown table, and highlight assumptions, constraints, and potential cost drivers that could affect the final figures.

Cost Item	**Estimate**	**Type**
Infrastructure & Cloud	$15,000	One-time / Initial
Software Licenses (Annual)	$5,000	Recurring / Ongoing
External Consultancy	$20,000	One-time / Initial

Table 3.4: Project Cost Breakdown (Example)

A clear and transparent cost analysis enables decision-makers to balance the required investment against the anticipated benefits, supporting informed go/no-go decisions.

With a clear vision and objectives established, it becomes essential to examine the financial implications. A realistic cost assessment ensures the project is both feasible and justifiable.

3.8 Risks

A thorough risk assessment is essential for anticipating potential issues that may affect the success of the project. This section should include a high-level risk overview, outline key assumptions, and identify any relevant regulatory or compliance factors.

3.8.1 Risk Identification

Highlight major risks related to:

- **Scope creep**: uncontrolled changes or continuous growth in project scope.
- **Resource constraints**: limited staff, expertise, budget or time.

- **Technical feasibility**: reliance on untested technologies, complex integrations, or outdated systems.
- **Stakeholder alignment**: conflicting interests, lack of engagement, or unclear ownership.
- **Dependencies and external factors**: third-party delays, market volatility, or legal changes.

3.8.2 Assumptions

Explicitly state the assumptions on which the analysis and proposed solution rely, for example:

- Availability and commitment of stakeholders.
- Budget approval and funding stability.
- Stability of requirements over the project lifecycle.
- Availability and reliability of key technologies.

Assumptions should be tracked throughout the project, as their invalidation can transform into new risks.

3.8.3 Regulatory and Compliance Considerations

Document all applicable laws, regulations, and organizational policies (e.g., GDPR, data retention requirements, accessibility standards). Failure to comply can create significant legal, financial, or reputational risks.

3.8.4 Risk Mitigation Strategy

For each identified risk, define:

- **Likelihood**: low, medium, high.
- **Impact**: minor, moderate, critical.
- **Mitigation measures**: e.g., contingency plans, phased validation, additional stakeholder engagement or fallback strategies.

Once risks have been identified and assessed, the Business Analyst should define how they will be addressed. In practice, there are four main approaches:

Avoidance: modify the project plan or approach to completely eliminate the risk. For example, if a new technology is too unstable, the project team may decide to avoid using it altogether.

Reduction (Mitigation): proactively reduce either the likelihood of the risk occurring or its potential impact. This may involve early prototyping, additional training, or phased rollouts.

Transfer (Sharing): shift the responsibility of the risk to a third party, often through contracts, outsourcing, or insurance. For example, relying on a cloud service provider with guaranteed service-level agreements (SLAs).

Acceptance: acknowledge the risk without taking immediate action beyond monitoring it. This is usually appropriate when the cost of addressing the risk exceeds its potential impact, or when the impact is minor.

In real projects, these strategies are often combined. For example, critical risks may be avoided, operational risks reduced, financial risks transferred, and minor risks accepted. The important point is to document and communicate the chosen approach, so that stakeholders understand both the potential threats and the agreed strategy for handling them.

3.9 Business Requirements

The list of Business Requirements can either be a chapter within the initiation document or a standalone document. The purpose of business requirements is to define the underlying business need of a project and establish the criteria for its success. These requirements describe *why* the project is needed, *whom* it will benefit, *when* and *where* it will take place, and which standards will be used to evaluate its success.

Importantly, business requirements do not define *how* the project will be implemented. They are focused on the "what" and "why", leaving the "how" to solution and technical design.

3.9.1 Distinguishing Needs from Requirements

It is crucial to distinguish between a business need and a business requirement:

- **Business Need**: "The modification is needed due to several inefficiencies identified over the past four years."
- **Business Requirement**: "The new solution should address these inefficiencies by ensuring tasks are automatically assigned to the most suitable person, improving overall efficiency."

To write effective business requirements, two techniques are particularly useful:

- **Business Epics**: large user stories that need to be broken down into smaller items.

- **The Verb + Noun pattern**: e.g., *Manage Users*, *Create Reports*.

Furthermore, good business requirements usually incorporate **Nouns** (to support data modelling), **Verbs** (to support process modelling), and **Rules** (to capture business logic).

Requirement Traceability Matrix

Organizing business requirements in a structured table ensures clarity and traceability. *Note: FR = Functional Requirements, NFR = Non-Functional Requirements.*

ID	Title	Description (Business Epic)	Priority	Why	Owner	Related Req.
BR1	Online booking	As a patient, I want to schedule medical appointments online...	Must Have	Efficiency	Dr. Smith	FR1, FR2, NFR1, NFR2
BR2	Auto invoice	As a finance officer, I want invoices to be generated automatically...	Should Have	Accuracy	J. Doe	FR3
BR3	Onboarding checklist	As an HR manager, I want a digital onboarding checklist...	Should Have	Compliance	L. Brown	FR4

Table 3.5: Requirement Traceability Table

The **Related Requirements** column ensures bidirectional traceability. This enables the team to answer:

- **Top-down**: "Why and how was this implemented?"
- **Bottom-up**: "What business need does this functionality serve?"

3.9.2 Business Epic Analysis

In Business Epic analysis, verbs identify processes, while nouns help identify data entities. Consider the statement: *"As a traveller, I want to book a hotel room to stay during my holiday."*

- **Verbs (Processes)**: "book" suggests steps like *Search rooms*, *Book*, *Check-in/out*.
- **Nouns (Entities)**: "hotel room" and "traveller" indicate key data structures.

3.9.3 MoSCoW Prioritization

The MoSCoW method helps stakeholders agree on the relative importance of requirements:

Must Have: Essential requirements critical for project success.

Should Have: Important requirements that add significant value but aren't strictly essential for the first release.

Could Have: Desirable features that can be postponed if time or budget are limited.

Won't Have (this time): Agreed to be out of scope for the current release.

To maintain clarity, ensure that different versions of this prioritization are properly tracked, as *Should Have* requirements often evolve into new *Must Haves* over time.

3.10 Features Delivered for Each Release

For large-scale projects, it is essential to clearly define the features that will be delivered in each release or phase. Breaking down the project into manageable increments allows for better planning, tracking, and communication with stakeholders. Each release should have a well-defined set of functionalities that align with the overall business goals and priorities.

In addition to outlining the start and end dates of the entire project, it is crucial to establish high-level milestones. These milestones act as checkpoints, indicating when specific features or groups of features will be delivered and validated. This phased approach helps manage risks, incorporate feedback earlier, and ensures continuous value delivery.

Setting realistic deadlines for each release supports effective resource allocation and helps maintain project momentum. It also enables stakeholders to track progress against expectations and prepare for upcoming changes, such as user training or system integration. Ultimately, clear timelines and milestone definitions contribute to smoother project execution and greater transparency.

Phase / Release	**Start Date**	**End Date**	**Key Features**	**Milestone**
Project Initiation	May 1, 2025	May 15, 2025	Business Case, Stakeholder id.	Project kick-off, Approval
Requirements Gathering	May 16, 2025	June 15, 2025	Use cases, High-level scope	Requirements sign-off
Release 1: Core Module	June 16, 2025	Aug 15, 2025	Basic user management, auth.	Functional prototype
Release 2: Reporting	Aug 16, 2025	Oct 1, 2025	Dashboards, data export	UAT for Release 1
Project Closure	Dec 1, 2025	Dec 15, 2025	Documentation, training	Final sign-off, handover

Table 3.6: Release Plan and High-Level Implementation Schedule

While this structured release planning looks very effective on paper, in reality it is difficult to make highly accurate predictions—especially for long-term projects. This is why such plans must be reviewed regularly and adjusted as circumstances evolve. Nevertheless, release planning remains a critical discipline: sooner or later, stakeholders will expect it, so it is best to start applying it early and refine it continuously.

Release plans should never be treated as static. Review and adjust them at the end of each sprint or major release, and at least once per quarter for large initiatives. This ensures your road-map stays realistic, reflects shifting priorities, and continues to deliver measurable value.

With a solid understanding of the "Why", the next step is to explore the "What"— defining in detail the business requirements that will drive the project forward.

Chapter 4

The What and How questions, step 2

At this stage, we move into the detailed scope of the solution by addressing two fundamental questions:

- **What** are the user's goals?
- **How** will those goals be achieved?

It is common for the "What" and the "How" to be described together within the Use Cases. However, it is important to distinguish them: the **What** refers to the user's goals and the objectives identified in the Business Case, while the **How** defines the specific path to achieving them.

> *Best practice: The Business Analyst should prepare one document per Use Case to ensure clarity and maintain traceability.*

To conduct a complete analysis, the following core elements should be examined in detail:

1. **Who are the actors?** This also implicitly defines the users. Compare with those identified in the initiation stage to ensure consistency.
2. **What are their goals in using the system?** Each Use Case represents one of these goals.
3. **How will the actors interact with the system to achieve these goals?** This step defines the solution requirements, which are categorized into **Functional Requirements Non-Functional Requirements**.

Functional requirements describe the user's activities needed to achieve a specific goal. They define the specific tasks or actions the solution must support.

On their own, however, functional requirements often appear as isolated statements, making them harder to interpret in context.

This is where **Use Cases** provide essential structure. A Use Case bundles together the relevant functional requirements and arranges them into a logical flow of steps, showing how a user (actor) and the system collaborate to achieve a goal.

> *Best Practice: supplement your Use Cases with **Wireframes**. They provide excellent support for both development and stakeholder discussions—after all, a picture is worth a thousand words.*

To sum up: functional requirements are the steps to achieve a goal, and they should be described as use case scenario steps, ideally complemented with visual mock-ups for clarity and validation.

4.1 Use Cases: The Core Tool

At this stage, the Business Analyst must translate high-level objectives into actionable requirements. The most effective tool for this is the **Use Case**, a structured way of describing how users (actors) interact with the system to achieve specific goals.

A Use Case must be **goal-oriented**: it captures what the user wants to achieve and how the system supports that goal. Well-written Use Cases are also user-centric, keeping the user experience at the centre of analysis. It is a fundamental rule that each Use Case must represent one specific goal for a particular actor.

> *Since every Use Case requires a title, I normally use the **User Story** format ("As a [user], I want to [action], so that [benefit]") as the title. This immediately clarifies the intent and the value of the requirement. In other scenarios, a User Story is an excellent way to summarise a complex Use Case that, for contextual reasons, may have been documented more formally elsewhere.*

4.1.1 Briefly described Use Case

Lightweight descriptions may be sufficient for small projects or early-stage analysis, such as brainstorming sessions. This format provides just enough information to understand the intent without getting bogged down in detail. It usually includes:

- **Actor**: The role that interacts with the system.
- **Use Case Name**: A concise goal description (e.g., *Book Course* or User Story template).
- **Short Description**: A one- or two-sentence summary of what the actor achieves.

Example:

- **Actor**: Registered Customer
- **Use Case Name**: "Book Course" or "As a registered customer I want to [...]"
- **Short Description**: This use case allows registered customers to book course places through the website.

4.1.2 Fully described Use Case

A fully described use case is recommended for large, complex, or high-risk projects. It includes the following elements.

Field	Description
ID	Unique identifier (e.g., UC1).
Actor	The primary user or role (e.g., Registered Customer).
Use Case Name	Goal expressed as a short title (e.g., Book Course).
Brief Description	A short explanation of what the Use Case covers.
Goal / Name	The intended outcome for the actor.
Scope	The system or subsystem in which the Use Case resides.
Preconditions	Conditions that must be true before the Use Case starts.
Success Guarantees	What the system guarantees if the Use Case completes successfully.
Main Flow	The "happy path": the standard sequence of steps (Functional Requirements).
Alternate Flows	Variations that still lead to the goal (optional paths).
Exception Flows	Scenarios where errors occur; how the system handles them.
Postconditions	What is true after the Use Case ends.
Acceptance Criteria	Conditions for validating success (e.g., Gherkin syntax).
Non-Functional Req.	Constraints such as performance, security, or usability.
Business Rules	Rules that guide system behavior, written parametrically.

*The **Main Flow** is where the actual functional requirements live, described as a logical sequence of interactions between the Actor and the System.*

4.1.3 Use Case Scenario Steps

After the initial introduction of the fields, it is essential to describe the **use case scenario steps**: how users reach their goals, what specific actions they take, and how the system responds.

The traditional format follows a *"ping-pong"* sequence:

User action → System response → User continues...

Of course, variations in this flow are normal and depend on the specific interaction logic. For example:

`User` → `User` → `User` → `System` → `User` → `System` → `System`

Example of Main Flow

To ensure clarity, each step should be numbered and clearly identify who is performing the action:

1. **Actor:** Selects the "Book Course" option.
2. **System:** Displays the list of available courses and dates.
3. **Actor:** Chooses a specific course and clicks "Confirm Selection".
4. **System:** Validates availability and displays the payment summary.
5. **Actor:** Enters payment details and submits the form.
6. **System:** Processes the transaction, sends a confirmation email, and displays a success message.

4.1.4 Visual Support

I recommend integrating wireframes directly into the "ping-pong" descriptions. When complemented by visual mockups, the Use Case becomes significantly more effective and intuitive. Visuals assist in conceptualizing the flow, improving memory retention, and—most importantly—providing a **rigorous feasibility check** for functional requirements.

1. **User** → selects a course from the first dropdown.
2. **System** → displays the list of quantities available for that specific course.
3. **User** → clicks the "Add to cart" button.

> *Reflecting on professional practice: While much of the literature [1] focuses on structured textual descriptions, my experience has taught me that visual mockups are far more effective than text alone—after all, a picture is worth a thousand words.*

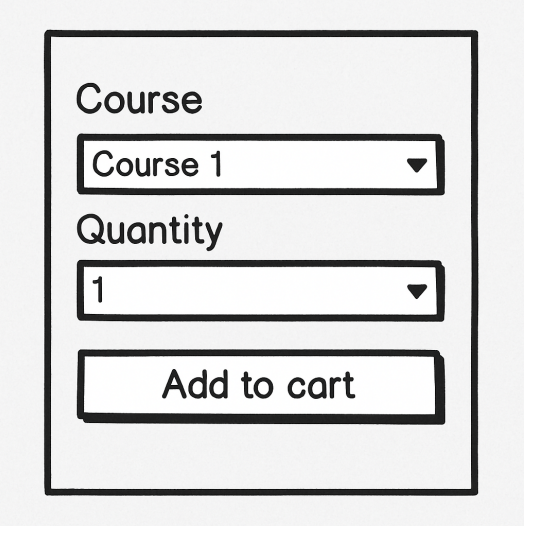

Figure 4.1: Wireframe example for the 'Book Course' use case.

4.2 Main Flow, Alternate Flows, and Exceptions

When describing a Use Case, it is important to distinguish between three different kinds of scenarios: the Main Flow, the Alternate Flows, and the Exception Flows.

4.2.1 The Three Flow Types

Main Flow (aka the "Happy Path"): This is the standard sequence of steps where everything goes as expected and the actor successfully reaches the goal. It usually follows a simple "ping-pong" structure:

$$\textit{User action} \rightarrow \textit{System response} \rightarrow \textit{User action} \rightarrow \ldots$$

Alternate Flows: These are variations in the interaction that still lead to the same goal, but through a different path. For example, a user might choose to pay via PayPal instead of a credit card. They are often considered "nice-to-have" and, depending on project priorities, their implementation can be postponed without blocking the core solution.

Exception Flows: These describe what happens when something goes wrong. They capture error conditions, unexpected inputs, or system failures. A system that cannot handle exceptions gracefully is incomplete and will fail in a production environment.

As a **rule of thumb**, the Business Analyst and the development team should prioritize as follows:

- Main Flows **must always** be implemented.
- Exception Flows **must always** be implemented to ensure system reliability and proper error handling.
- Alternate Flows **can be considered optional**, as the user can achieve the goal through the Main Flow.

Even when using *briefly described* Use Cases, the Business Analyst must still ensure the inclusion of:

- The Main Flow.
- Exception flows (error conditions).
- Acceptance criteria.
- Business rules.
- Non-functional requirements defined in a testable way.

4.3 Business Rules in Parametric Form

Alongside the interaction flows, the Business Analyst must define the relevant Business Rules. A best practice is to write these in a **parametric way**, using named parameters instead of hard-coded values. This approach makes rules easier to maintain, test, and adapt as business needs change, often without requiring a full code deployment.

Example:

> **Scenario Definition:** When the user books a course, the system must verify that the maximum number of participants (**BR1**) has not been exceeded and that the booking deadline (**BR2**) has not passed.

- **BR1:** `Maximum_Participants = 30`
- **BR2:** `Booking_Deadline_Hours = 24`

> **System Logic:** If either rule is not satisfied, the system prevents the booking and notifies the user with a specific message.

Using this parametric approach allows the business to change the `Maximum_Participants` or `Booking_Deadline_Hours` values in a configuration file or a database table, ensuring the system remains flexible without the need for a new software release.

4.4 Use Case Example

Use Case ID	UC1
Actor	Registered Customer
Use Case Name	Book Course
Short Description	This use case allows registered customers to book places in a course through the website.
Business Requirements	Req. 1
Priority	Must-have
Preconditions	1. The user is authenticated (logged in). 2. Courses are available in the catalogue.
Main Flow	1. The user selects a course from the dropdown list. 2. The system displays course details (title, date, available seats). 3. The user clicks "Book Course." 4. The System validates business rules (BusRule1 and BusRule2). 5. The System confirms the booking and displays confirmation details.
BusRule1	Maximum number of participants per course = 30
BusRule2	Booking deadline = 24 hours before course start
Exception EF1	**Course Full:** The system displays a message: "This course is fully booked. Please choose another."
Exception EF2	**Deadline Passed:** The system displays a message: "Booking for this course is closed."
Acceptance Criteria Scenario 1	**Given** the user is logged in, **And** the course has available seats, **And** the booking deadline has not passed, **When** the user clicks "Book Course", **Then** the booking is confirmed.
Acceptance Criteria Scenario 2	**Given** the course has no available seats, **When** the user clicks "Book Course", **Then** the system displays an error message.
Acceptance Criteria Scenario 3	**Given** the booking deadline has passed, **When** the user clicks "Book Course", **Then** the system displays an error message.

Note on Gherkin Syntax: The Acceptance Criteria above follow the Gherkin format, a structured language that uses *Given-When-Then* statements to define system behavior. This syntax bridges the gap between business requirements and technical testing, ensuring that scenarios are easily understood by stakeholders and directly transformable into automated test scripts.

4.5 Non-functional Requirements

Alongside functional requirements, each Use Case should also include the relevant **Non-functional Requirements (NFRs)**. While functional requirements describe *what* the system must do, non-functional requirements describe the *qualities* the system must possess while doing it.

Typical categories of non-functional requirements include:

- **Performance**: Response times, throughput, and acceptable load levels.
- **Usability**: Ease of use, accessibility standards, and user satisfaction.
- **Security**: Identification, Authentication, Authorization, Auditing, Accountability, and Non-Repudiation.
- **Privacy**: Compliance with regulations (e.g., GDPR), data minimization, and retention.
- **Scalability**: The maximum workload the system must sustain while meeting performance targets.
- **User Interface details**: Aspects such as the layout, which should be documented as prototypes rather than embedded in the narrative. *Wireframes* are simple sketches for discussion; *Prototypes* are interactive, higher-fidelity models used to validate usability before development.

4.5.1 Examples of Testable NFRs

To be effective, NFRs must be defined in a measurable and testable way. Remember that you can use the parametric form here as well.

- **Performance**: The system must load the available courses list and details in under **1.5 seconds** under a concurrent load of 500 users.
- **Security**: Transfers above €10,000 must require **two-factor authentication**.
- **Privacy**: Personal data of participants must be automatically **purged 30 days after** the course completion, unless consent is renewed.
- **Accountability**: The system must log every booking attempt, including the **timestamp, User ID, and IP address**, to provide a full audit trail in case of disputes.
- **Availability**: The booking service must have an up-time of **99.9%**during business hours (08:00–20:00), ensuring that users can always access the catalog.

Reflecting on professional practice: Unfortunately, NFRs are often overlooked during the analysis phase. In my experience, two classes are almost always forgotten: ***security requirements*** *and* ***performance requirements****. Neglecting these can undermine an otherwise well-designed solution.*

4.5.2 Security Requirements

While security is an expansive subject, this chapter focuses on the core essentials every Business Analyst must address to ensure a secure solution.

Many stakeholders mistakenly believe that security only concerns techy people or defending against external attacks. In reality, every system manages sensitive data, and the first line of defense is **access control**: identifying who can access specific data and defining their respective rights.

The CIA Triad

The classical **CIA triad** should always be addressed when defining security requirements:

- **Confidentiality**: Information is accessible only to authorized individuals.
- **Integrity**: Information remains accurate and unaltered without permission.
- **Availability**: Information is available in a timely manner, even under stress. This includes backups, recovery procedures, and disaster recovery planning.

Depending on the project's sensitivity, **non-repudiation** (ensuring a party cannot deny an action) and **accountability** (traceability of actions) should also be considered. There is also one golden rule that must always be remembered: **every user should be granted only the minimum access rights necessary to perform their job**—nothing more, nothing less.

The Business Analyst does not need to design technical security controls, but must ensure that security requirements are clearly captured, validated with stakeholders, and defined in a **testable** manner.

Furthermore, throughout the product life-cycle, it is crucial to remember that security and technology management are deeply intertwined. This requires both the team and the system to be prepared for regular updates. While this may sound straightforward, it is often complex. For instance, how can you ensure that the system still performs as expected after upgrading to a new library? This is why a test environment, as close to the production environment as possible, is essential to validate the system before any live deployment.

Strategic Questions for Security Requirements Elicitation

A practical starting point for defining security-related NFRs—especially if you do not have a cybersecurity background—is to ask the following questions during stakeholder interviews:

- Who has access to the application?
- What would motivate someone to attack this application?
- What specific targets (data or functions) are exposed?

- How might each target be attacked? (Consider frameworks such as *STRIDE*, *DREAD*, or *PASTA*).
- What is the business impact of a successful attack?
- If a critical vulnerability is identified, what is the defined Service Level Agreement (SLA) for remediation?

  A Note on Risk Management: Remember that **Risk = Probability × Impact**. When discussing mitigation, always evaluate whether prevention, detection, and response is the most cost-effective strategy for that specific business context.

4.5.3 Global Non-Functional Requirements

While many NFRs are specific to a single Use Case, some requirements relate to the system as a whole (e.g., global security protocols or general platform availability). In these cases, they should not be repeated in every Use Case; instead, they should be documented in a separate, centralized document to ensure consistency across the entire solution.

4.5.4 Prioritizing Non-Functional Requirements

Not all NFRs have the same impact on business value or implementation effort. A pragmatic way to prioritize them is:

- **High priority**: High business value and low implementation difficulty.
- **Medium priority**: Features with average value/effort. These should be prioritized based on expected frequency of use and time-to-market constraints.
- **Low priority**: Low value and high difficulty. These are often postponed to future releases.

Medium-priority requirements can be promoted to high priority if they are frequently used or critical to user adoption. Otherwise, if resources are tight, they typically move toward the Low-priority list.

> *Reflecting on professional practice: While I prefer the MoSCoW approach for functional requirements, I find that a simple High-Medium-Low scale is often more effective and intuitive for NFRs.*

4.5.5 Example: NFRs for Course Booking

In this scenario, while functional requirements define the flow (select → confirm → receive), the NFRs define the quality and security constraints of that experience.

Category	Requirement	Priority
Performance	The system must support at least 100 concurrent users with a response time $\leq$ 2 seconds.	High
Security	Only authenticated users can book. All communications must use TLS 1.3 encryption.	High
Privacy	User personal data must be stored in compliance with GDPR with explicit consent.	High
Availability	The booking service must guarantee 99.5% uptime per month, excluding maintenance.	Medium
Accessibility	The booking page must comply with WCAG 2.1 Level AA standards.	Medium
Scalability	The system must handle a 50% increase in bookings (up to 150 users) without degradation.	Low

Table 4.3: Prioritized NFRs for the Course Booking use case.

4.5.6 Key Insight for the BA

⇒ NFRs don't describe *what* the system does (that's functional), but *how well* it must do it.

⇒ NFRs must be expressed in **measurable, testable terms** (e.g., "response time $\leq$ 2s", "uptime $\geq$ 99.5%").

⇒ NFRs' prioritization ensures stakeholders understand the necessary **trade-offs**.

Chapter 5

Change Requests

Change Management involves analyzing the effect of a proposed change on project requirements or managing the transition from the current state (*As-Is*) to the future state (*To-Be*).

While there is no need to react to every Change Request immediately, the priority must be to analyze the request and its potential impact. Only after this assessment can a decision be made to either modify the current work or maintain the original plan, deferring the change to a future release.

5.1 The Change Control Process

To manage this effectively, a formal method to receive and log requests must be established. This log allows the team to analyze each request and determine its impact on the work in progress.

For changes with a major impact on the project's scope, budget, or timeline, approval must be granted by a dedicated group of stakeholders. This group is typically known as the **Change Control Board (CCB)** or the **Steering Committee**.

5.2 Best Practices for Change Management

Managing changes requires discipline and transparency. Follow these core principles:

Log and Analyze:
Every request must be documented before any action is taken. Never jump into implementation without understanding the ripple effects on other requirements.

Version Control:
Always use version control for all documentation and requirements to track *what* changed, *when*, and *why*. This provides a crucial audit trail for the project.

Communication:
Once a change is approved, ensure that every stakeholder and team member is made aware of the update. Transparency prevents misalignment and redundant work.

Validation:

Always remember that every Use Case must be validated by the relevant stakeholder before it is considered final.

Focus for the BA: Your role is not just to record changes, but to act as a gatekeeper who ensures that every modification adds genuine value without compromising the project's integrity.

About the Author

The author graduated in Computer Science from University of Bologna and he is a seasoned Cybersecurity Specialist with over two decades of experience across the private sector, academia, and European Union institutions. He currently serves as an **Information Technology Security Specialist** at the **European Union Intellectual Property Office (EUIPO)** in Alicante, Spain, where he leverages a multidisciplinary background to protect critical digital infrastructure.

Throughout his career, he has held pivotal roles within the European landscape. He served as the **Vulnerability Management Lead** for the **European Parliament**, ensuring the resilience of legislative systems against emerging threats. Prior to that, he spent several years at the **European Commission**, specifically within the **European Anti-Fraud Office (OLAF)** in Brussels, where he acted as a Senior Functional and Business Analyst, streamlining the digital processes essential to protecting the Union's financial interests.

His technical expertise is broad, encompassing Software Development, Cybersecurity, and Business Analysis. His earlier professional journey includes a period as a **Computer Science Teacher** in Italy, as well as leadership roles in software development and Cybersecurity consulting, including significant tenures at companies such as *Communication Valley Reply* and *C-Global (Cedacri)*.

The author holds a strong academic and practical foundation in:

- **Cybersecurity:** Penetration testing, ethical hacking, and real-time security monitoring.
- **Software Engineering:** Architecture design, discrete event simulations, and agile project management.
- **Functional Analysis:** Requirements engineering and business process optimization.

He continues to live and work in Alicante, dedicated to the mission of securing the intellectual and digital assets of the European Union.

Bibliography

[1] Yayici, E. *Business Analyst's Mentor Book: With Best Practice Business Analysis Techniques and Software Requirements Management Tips.*, 2012.